PIC μC implements CRC-16 algorithm

Lon Glastner, Solutions Cubed, Chilo, CA

Detecting errors in serial data can be paramount in completing an embedded-control design. Determining which algorithm to use for detecting serial-communications errors depends on several factors. Ideally, the method should require minimal hardware and little computational power from your processor and still provide high-level protection against undetected data errors. The cyclic redundancy check (CRC) combines all these factors under one umbrella. A multitude of CRC flavors exists, including the Dow CRC (8 bits), CRC-16, and CRC-CCITT (both 16 bits). The CRC-16 uses a 16-bit shift register and can detect the following error types:

- any cluster of errors within a 16-bit section of data,
- any odd number of errors within the data field,
- all double-bit errors in the data field, and
- most large clusters of errors.

You can characterize the CRC-16 as both a polynomial expression and as a hardware-based shift register (Figure 1). You can implement a CRC-16 in a midrange PIC μC with minimal coding and without additional hardware. Selecting a μC with an on-chip USART, such as the PIC16C63, eases serial communications. This implementation does not focus on the mathematical proof of a CRC-16's error-correcting effectiveness. The CRC algorithm is so effective it's an industry-accepted method for detecting data errors. The heart of a CRC-16 algorithm is a shift register. You generate the shift register by shifting each data bit through the algorithm. In this implementation, the data shifts the most significant bit first, one data byte at a time.

Two temporary registers buffer the data to prevent the shifting from corrupting the data. The shift register comprises two separate 8-bit registers, CRC16_HI and CRC16_LO. The most significant bit of CRC16_HI is the location of Stage 16 (Figure 1). From the figure and Table 1, you can see that the result of XORing the input data bit and the contents of Stage 16 of the shift register determines the effect that new data has on the shift register. If the result is a one, then you must complement

Table 1—XOR truth table		
X	Y	XOR
0	0	0
0	1	1
1	0	1
1	1	0

the contents of stages 2 and 15 before rotating the new data into the shift register. If the result is a zero, then the new data can rotate immediately into the shift register. Some housekeeping tips can be helpful here. The data transmitter should generate its CRC-16 in the same manner as the data receiver. Also, it's advisable to clean the CRC-16 shift register before rotating the first data bit of the data string into it.

How does the CRC-16 identify data errors? The simplest method is to attach the shift register to the end of the data string. In this implementation, the CRC16_HI register should follow the last byte of data sent. The CRC16_LO register follows the CRC16_HI register. If the receiving system computes a CRC-16 value from all the data bytes and the attached shift register (CRC16_HI and CRC16_LO), then the resulting CRC-16 code is 0000h. Any nonzero result indicates an error in the data. In some systems, an error may occur that results in all zeros being sent as the data and attached CRC-16. This type of error poses as error-free data. In these systems, you can overcome the false indication by complementing the CRC-16 before attaching it to the data string. The CRC-16 shift register generated by attaching the complement is always 800Dh.

The code fragment in Listing 1 generates a CRC-16 shift register on a byte-by-byte basis. You could embed this code within serial-receive and serial-send routines to

provide a powerful error-detection tool. You can easily generate the CRC-16 on the fly, thus minimizing the use of processor resources (DI #2321).

```
;********************************************************************
;MAKE_CRC16 - This routine generates a CRC-16 shift register from the data byte sto
;in the DATA_REG register.  DATA_TEMP0 and DATA_TEMP1 are used to buffer the data
;and prevent it from being corrupted by the CRC-16 generation process.
;********************************************************************
;
MAKE_CRC16
        movf    DATA_REG,W
        movwf   DATA_TEMP0                      ;Store data in temporary register
        movlw   '00001000'b                     ;Set counter for 8 data bits
        movwf   NUMBER_BITS                     ;Load counter register
More_Rotates
        movf    DATA_TEMP0,W            ;Move buffered data to 2nd buffer
        movwf   DATA_TEMP1                      ;This register is corrupted with ev
        movf    CRC16_HI,W                      ;Move upper shift register to worki
        xorwf   DATA_TEMP1                      ;XOR shift register with data regis
        btfss   DATA_TEMP1,7           ;MSB is XOR of stage16 and input data bit
        goto    No_Xorwf                        ;If bit is clear then no complement
        movlw   '00000010'b                     ;Prepare to complement stage2
        xorwf   CRC16_LO                        ;Complement stage2 of shift registe
        movlw   '01000000'b                     ;Prepare to complement stage15
        xorwf   CRC16_HI                        ;Complement stage15 of shift regist
No_Xorwf
        rlf     DATA_TEMP0                      ;Rotate next data bit into position
        rlf     DATA_TEMP1                      ;Rotate XOR of input into CRC16_LO
        rlf     CRC16_LO                        ;Shift CRC16 register
        rlf     CRC16_HI                        ;Shift CRC16 register
        decfsz  NUMBER_BITS                     ;Count out 8 data bits
        goto    More_Rotates                    ;Not finished with this data byte
        return                                  ;This byte is finished
```